1a) N-Queen

```
###########
########## 1a

def isSafe(board, row, col):

    # return false if two queens share the same column
    for i in range(row):
        if board[i][col] == 'Q':
            return False

    # return false if two queens share the same `\` diagonal
    (i, j) = (row, col)
    while i >= 0 and j >= 0:
        if board[i][j] == 'Q':
            return False
        i-=1
        j-=1

    # return false if two queens share the same `/` diagonal
    (i, j) = (row, col)
    while i >= 0 and j < len(board):
        if board[i][j] == 'Q':
            return False
        i-=1
        j+=1

    return True


def printSolution(board):
    for row in board:
        print(str(row).replace(',', '').replace('\'', ''))
    print()


def nQueen(board, row):

    # if `N` queens are placed successfully, print the solution
    if row == len(board):
        printSolution(board)
        return

    # place queen at every square in the current row `r`
    # and recur for each valid movement
    for col in range(len(board)):

        # if no two queens threaten each other
```

```
        if isSafe(board, row, col):
            # place queen on the current square
            board[row][col] = 'Q'

            # recur for the next row
            nQueen(board, row + 1)

            # backtrack and remove the queen from the current square
            board[row][col] = '–'
```

```
N = 4
board = [['–' for x in range(N)] for y in range(N)]
nQueen(board, 0)
```

```
[– Q – –]
[– – – Q]
[Q – – –]
[– – Q –]

[– – Q –]
[Q – – –]
[– – – Q]
[– Q – –]
```

1b) Water jug

```
# This function is used to initialize the
# dictionary elements with a default value.
from collections import defaultdict

# jug1 and jug2 contain the value
# for max capacity in respective jugs
# and aim is the amount of water to be measured.
jug1, jug2, aim = 4, 3, 2

# Initialize dictionary with
# default value as false.
visited = defaultdict(lambda: False)

# Recursive function which prints the
# intermediate steps to reach the final
# solution and return boolean value
# (True if solution is possible, otherwise False).
```

```python
# amt1 and amt2 are the amount of water present
# in both jugs at a certain point of time.
def waterJugSolver(amt1, amt2):

    # Checks for our goal and
    # returns true if achieved.
    if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
        print(amt1, amt2)
        return True

    # Checks if we have already visited the
    # combination or not. If not, then it proceeds further.
    if visited[(amt1, amt2)] == False:
        print(amt1, amt2)

        # Changes the boolean value of
        # the combination as it is visited.
        visited[(amt1, amt2)] = True

        # Check for all the 6 possibilities and
        # see if a solution is found in any one of them.
        return (waterJugSolver(0, amt2) or
                waterJugSolver(amt1, 0) or
                waterJugSolver(jug1, amt2) or
                waterJugSolver(amt1, jug2) or
                waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                amt2 - min(amt2, (jug1-amt1))) or
                waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                amt2 + min(amt1, (jug2-amt2))))

    # Return False if the combination is
    # already visited to avoid repetition otherwise
    # recursion will enter an infinite loop.
    else:
        return False

print("Steps: ")

# Call the function and pass the
# initial amount of water present in both jugs.
waterJugSolver(0, 0)
```

```
Steps:
0 0
4 0
4 3
0 3
3 0
3 3
```

```
4 2
0 2

True
```

1c) Best FS

```
###########
########### 1c

from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]



def printPQ(o,c):
    o = [x[1] for x in sorted(o)]
    print ("{:<25} {:<25}".format(str(o),str(c)))




def best_first_search(actual_Src, target, n):
    close=[]
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:

        u = pq.get()[1]
        close.append(u)

        if u == target:
            printPQ(pq.queue,close)
            print("Target Reached")
            return

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))

        printPQ(pq.queue,close)

    print("Target not reachable")
```

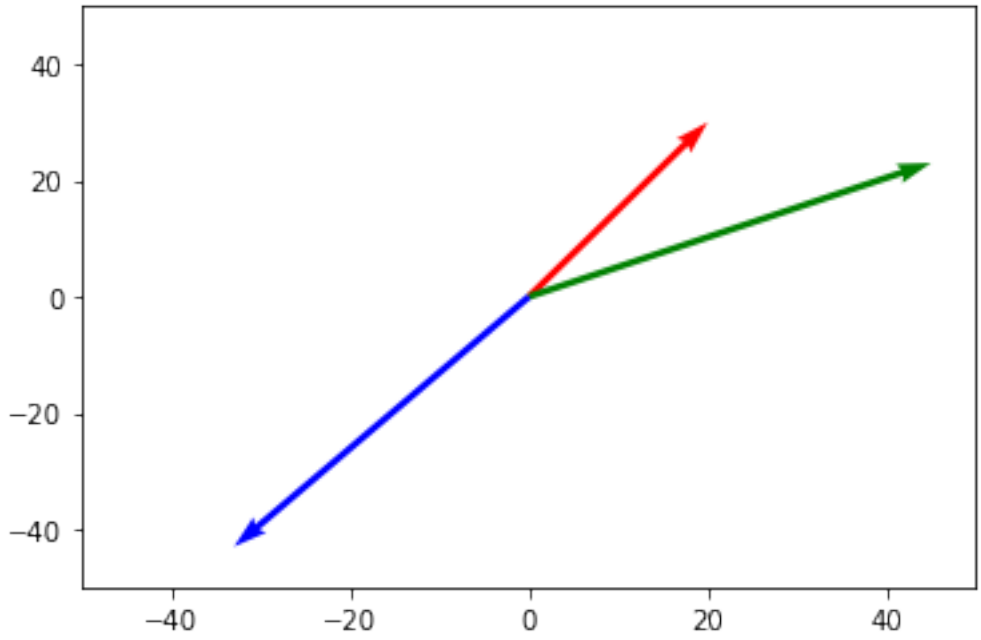
```
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))


# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
# addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
print ("{:<25} {:<25}".format('OPEN','CLOSE'))
best_first_search(source, target, v)
```

```
OPEN                      CLOSE
[1, 3, 2]                 [0]
[3, 2, 5, 4]              [0, 1]
[2, 8, 5, 4]              [0, 1, 3]
[8, 5, 4, 6, 7]           [0, 1, 3, 2]
[10, 5, 4, 6, 7]          [0, 1, 3, 2, 8]
[5, 4, 6, 7]              [0, 1, 3, 2, 8, 10]
[4, 6, 7]                 [0, 1, 3, 2, 8, 10, 5]
[6, 7]                    [0, 1, 3, 2, 8, 10, 5, 4]
[7]                       [0, 1, 3, 2, 8, 10, 5, 4, 6]
[]                        [0, 1, 3, 2, 8, 10, 5, 4, 6, 7]
Target not reachable
```

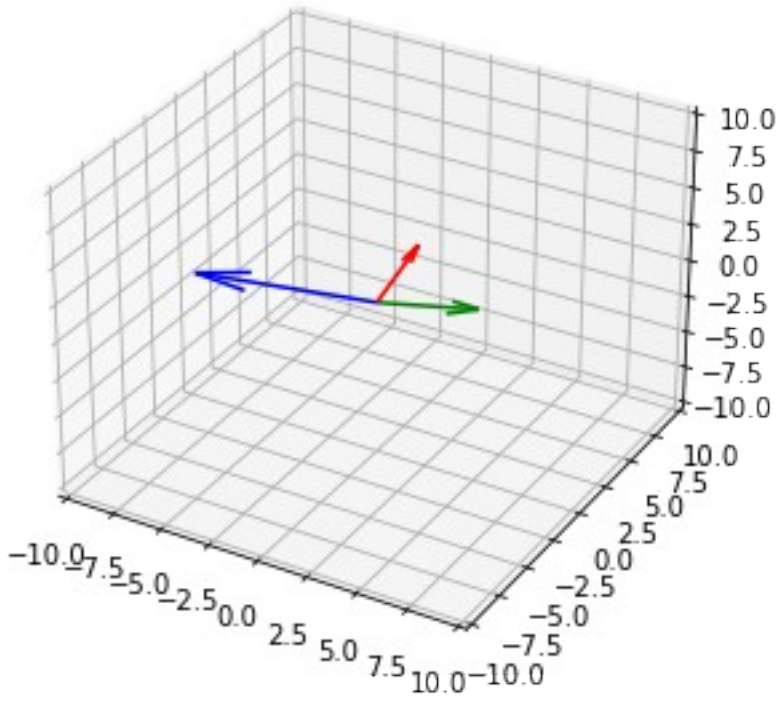
2) 2D & 3D

```
import numpy as np
import matplotlib.pyplot as plt
def plt2d(v,color):
 origin=[0,0]

plt.quiver(*origin,*v,color=color,angles='xy',scale_units='xy',scale=1
)
```

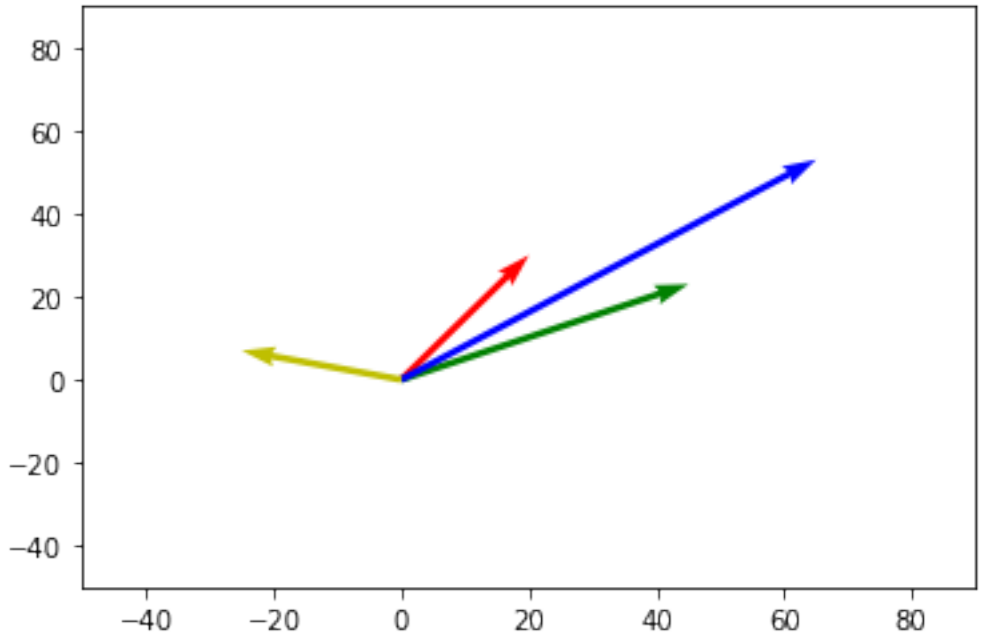
```
def plt3d(ax,v,color):
 ax.quiver(*[0,0,0],*v,color=color)

#2d
x=[20,30]
y=[45,23]
z=[-33,-43]
plt.xlim(-50,50)
plt.ylim(-50,50)
plt2d(x,'r')
plt2d(y,'g')
plt2d(z,'b')
```



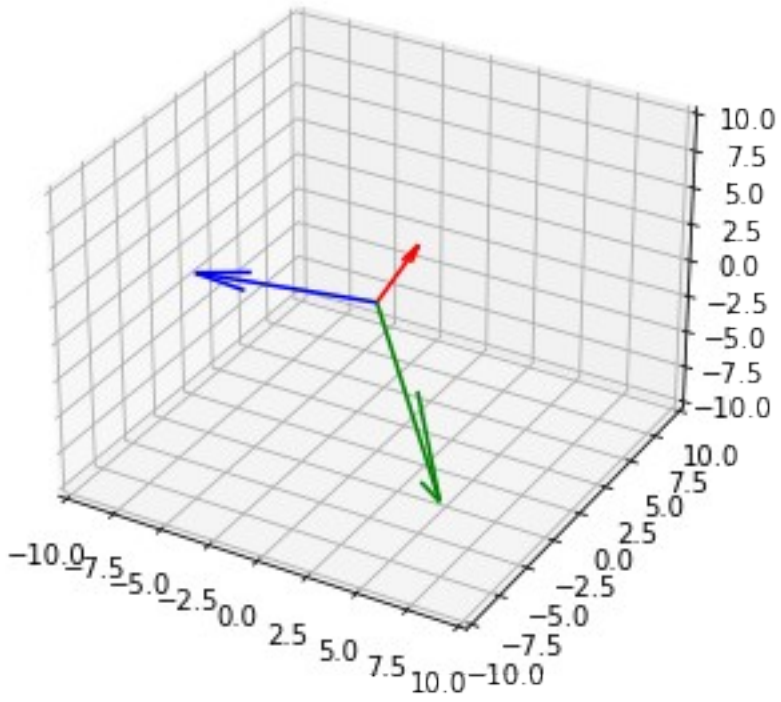

```
#3d
v1=[1, 2, 3]
v2=[-7, -4, 2]
v3=[3, 4, -2]
fig=plt.figure(figsize=(5,5))
ax=plt.axes(projection='3d')
plt.xlim([-10,10])
plt.ylim([-10,10])
ax.set_zlim([-10,10])
plt3d(ax,v1,'r')
plt3d(ax,v2,'b')
plt3d(ax,v3,'g')
```




```
#vector addition
x1=np.array([20,30])
y1=np.array([45,23])
plt.xlim([-50,90])
plt.ylim([-50,90])
plt2d(x1,'r')
plt2d(y1,'g')
plt2d(x1+y1,'b')
#subtraction
plt2d(x1-y1,'y')
```




```
#cross product
j1=[1, 2, 3]
j2=[-7, -4, 2]
fig=plt.figure(figsize=(5,5))
ax=plt.axes(projection='3d')
plt.xlim([-10,10])
plt.ylim([-10,10])
ax.set_zlim([-10,10])
plt3d(ax,j1,'r')
plt3d(ax,j2,'b')
plt3d(ax,np.cross(j1,j2),'g')
```

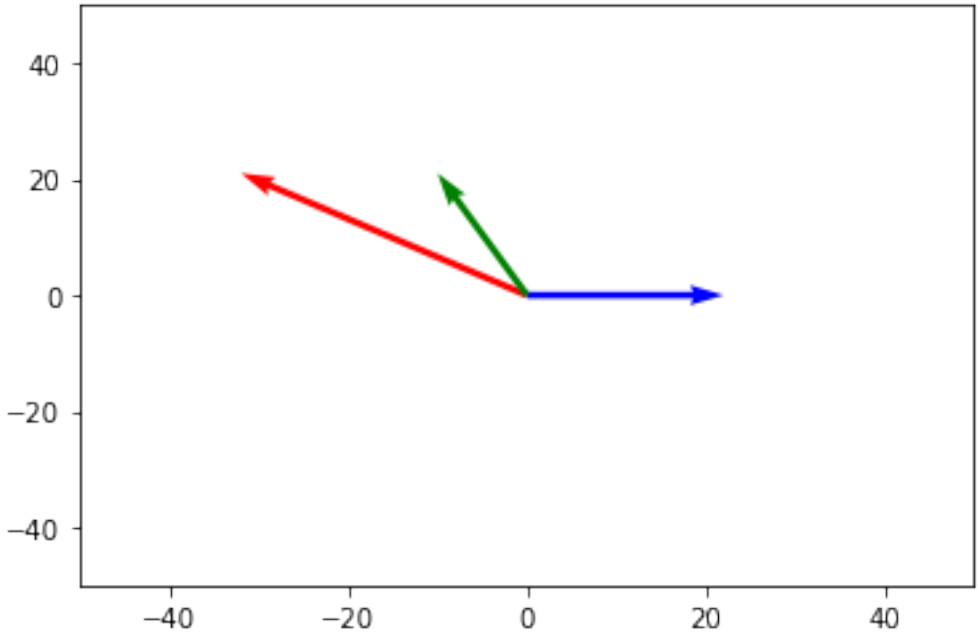


3) Inequalities

```
a=np.array([-32,21])
b=np.array([22,0])
maga=np.linalg.norm(a)
magb=np.linalg.norm(b)
dot=np.dot(a,b)
s=np.add(a,b)
if (maga*magb>=dot):
 print('sch')
else:
 print('no sch')
if (np.add(maga,magb)>=np.linalg.norm(s)):
 print('tri')
else:
 print('no tri')
plt.xlim([-50,50])
plt.ylim([-50,50])
plt.quiver(*[0,0],*a,angles='xy',scale=1,scale_units='xy',color='r')
plt.quiver(*[0,0],*b,angles='xy',scale=1,scale_units='xy',color='b')
plt.quiver(*[0,0],*(a+b),angles='xy',scale=1,scale_units='xy',color='g
')
```

```
sch
tri
```

```
<matplotlib.quiver.Quiver at 0x254efd04a30>
```

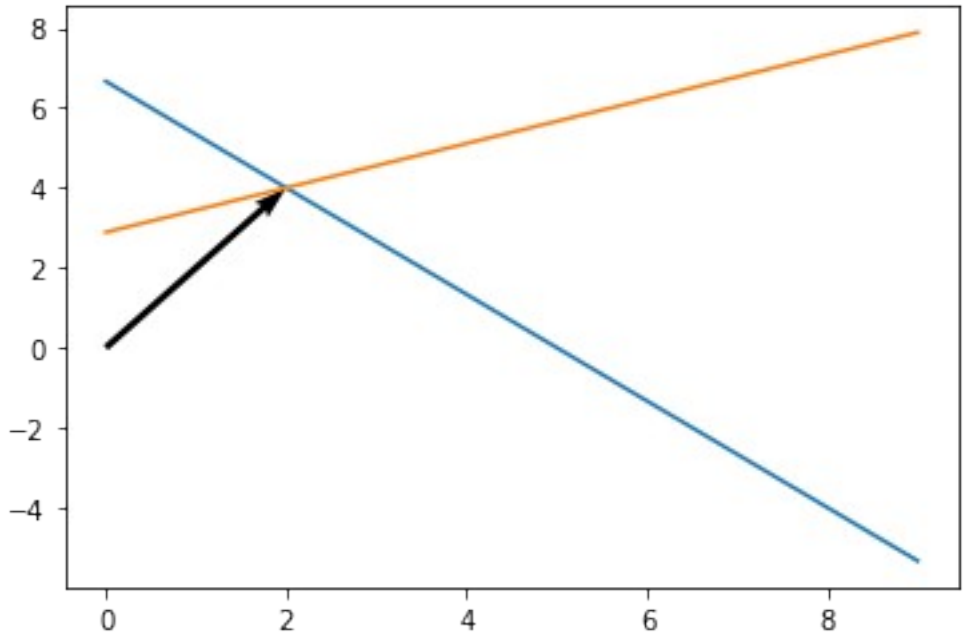


4) Ax=y

```
import matplotlib.pyplot as plt
import numpy as np
a=np.array([[4,3],[-5,9]])
b=np.array([20,26])
print('inverse=',np.linalg.inv(a))
e=np.linalg.solve(a,b)
print('the sol',e)
xp=np.linspace(0,9,1000)
y1=(20-4*xp)/3
y2=(26+5*xp)/9
plt.plot(xp,y1)
plt.plot(xp,y2)
plt.quiver(*[0,0],*[2,4],scale=1,scale_units="xy",angles='xy',units='x
y')
```

```
inverse= [[ 0.17647059 -0.05882353]
 [ 0.09803922  0.07843137]]
the sol [2. 4.]
```

```
<matplotlib.quiver.Quiver at 0x254eded9640>
```



8a) K -means

```
###########
########### 8a

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
j=sns.load_dataset('iris')
k=j.drop(['petal_length',"petal_width","species"],axis=1)

km=KMeans(n_clusters=3)
y=km.fit_predict(k)
```

```
C:\Users\Varun Kadya\anaconda3\lib\site-packages\sklearn\cluster\
_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

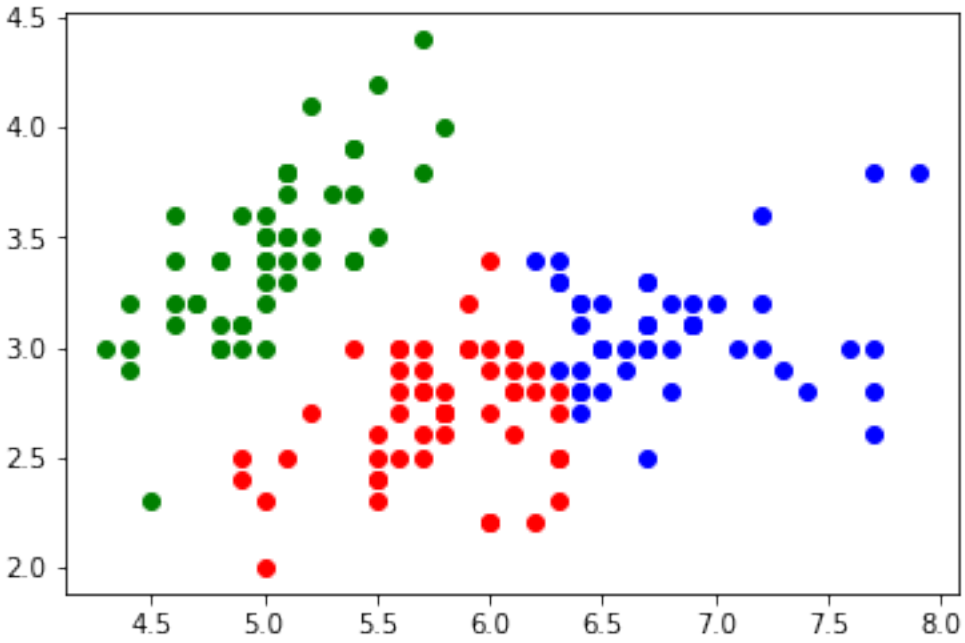
```
k['c']=y

print(k)
```

```
     sepal_length  sepal_width  c
0             5.1          3.5  2
```

```
1             4.9          3.0  2
2             4.7          3.2  2
3             4.6          3.1  2
4             5.0          3.6  2
..            ...          ... ..
145           6.7          3.0  0
146           6.3          2.5  1
147           6.5          3.0  0
148           6.2          3.4  0
149           5.9          3.0  1

[150 rows x 3 columns]
```

```
k1=k[k['c']==0]
k2=k[k['c']==1]
k3=k[k['c']==2]
plt.scatter(k1.sepal_length,k1.sepal_width,color='r')
plt.scatter(k2.sepal_length,k2.sepal_width,color='b')
plt.scatter(k3.sepal_length,k3.sepal_width,color='g')
```

```
<matplotlib.collections.PathCollection at 0x1ba7d0d9d90>
```



8b) EM Clustering

```
###########
########## 8b
```

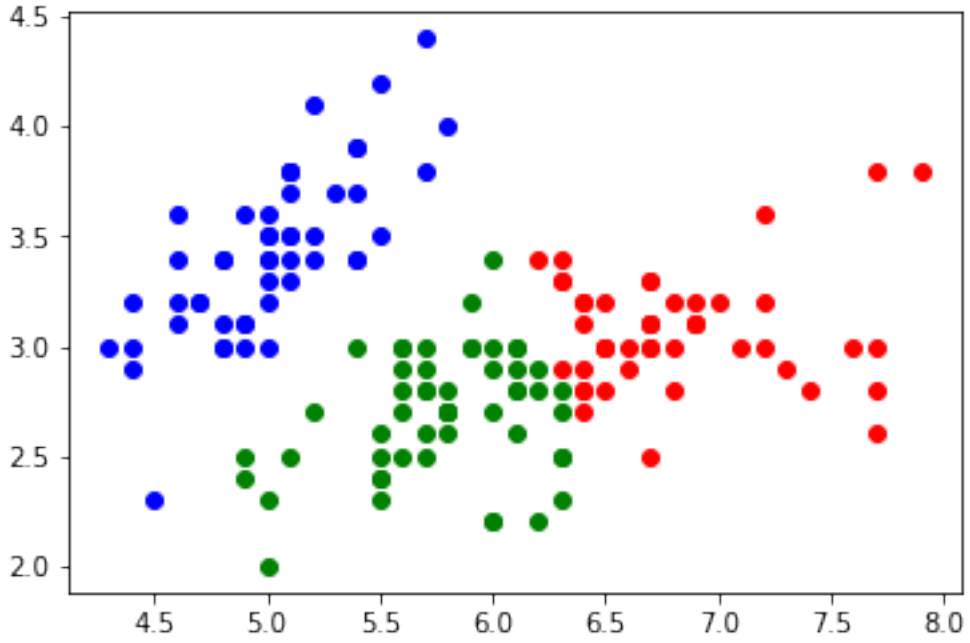
```
from sklearn.mixture import GaussianMixture as g
```

```
km1=g(n_components=3)
y1=km1.fit_predict(k)
```

```
C:\Users\Varun Kadya\anaconda3\lib\site-packages\sklearn\cluster\
_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

```
k['c1']=y1
k1=k[k['c1']==0]
k2=k[k['c1']==1]
k3=k[k['c1']==2]
plt.scatter(k1.sepal_length,k1.sepal_width,color='r')
plt.scatter(k2.sepal_length,k2.sepal_width,color='b')
plt.scatter(k3.sepal_length,k3.sepal_width,color='g')
```

```
<matplotlib.collections.PathCollection at 0x1ba7d681b50>
```

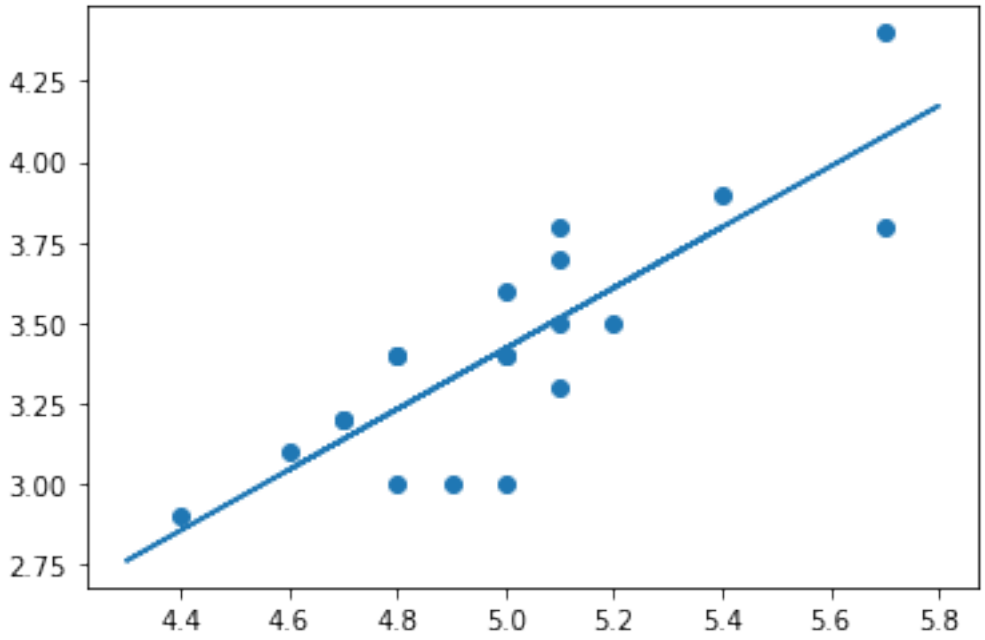


9a) Linear Regression

```
from sklearn import datasets,linear_model
import pandas as pd
import seaborn as sns
from sklearn.metrics import mean_squared_error,accuracy_score,r2_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

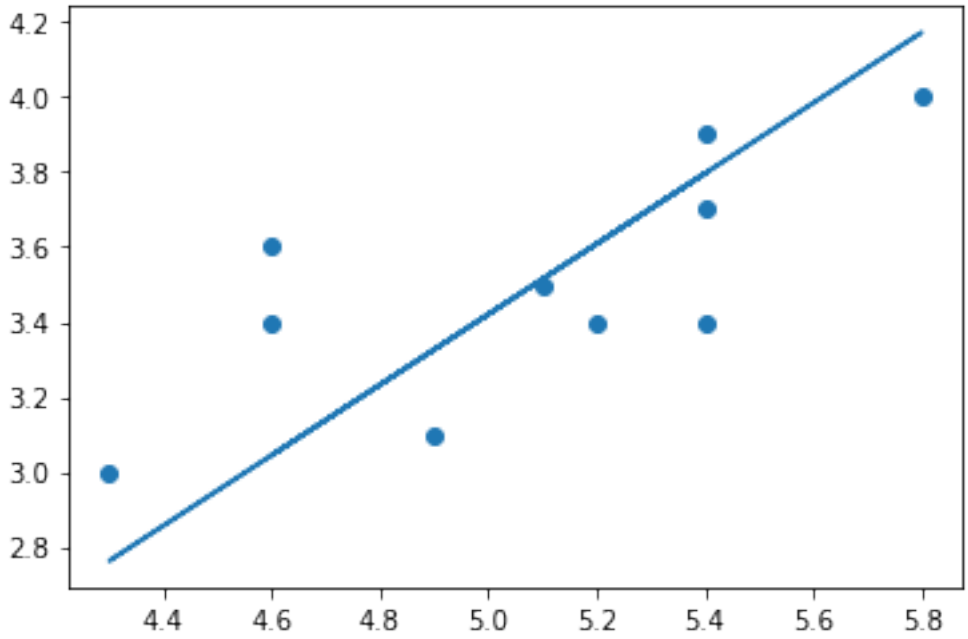
```
# k=sns.load_dataset('iris')
# X=k.iloc[:30,:1].values
# print(X)

k=sns.load_dataset('iris')
X=k.iloc[:30,:1].values
Y=k.iloc[:30,1:2].values
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=1/3)
model = linear_model.LinearRegression()


model.fit(x_train,y_train)
prediction = model.predict(x_test)

#print(r2_score(y_test,prediction))

#Training set
plt.scatter(x_train,y_train)
plt.plot(x_test,prediction)
```

```
[<matplotlib.lines.Line2D at 0x254edc57bb0>]
```




```
#Testing set
plt.scatter(x_test,y_test)
plt.plot(x_test,model.predict(x_test))
```

```
[<matplotlib.lines.Line2D at 0x254edca4850>]
```




```
print(r2_score(y_test,prediction))
```

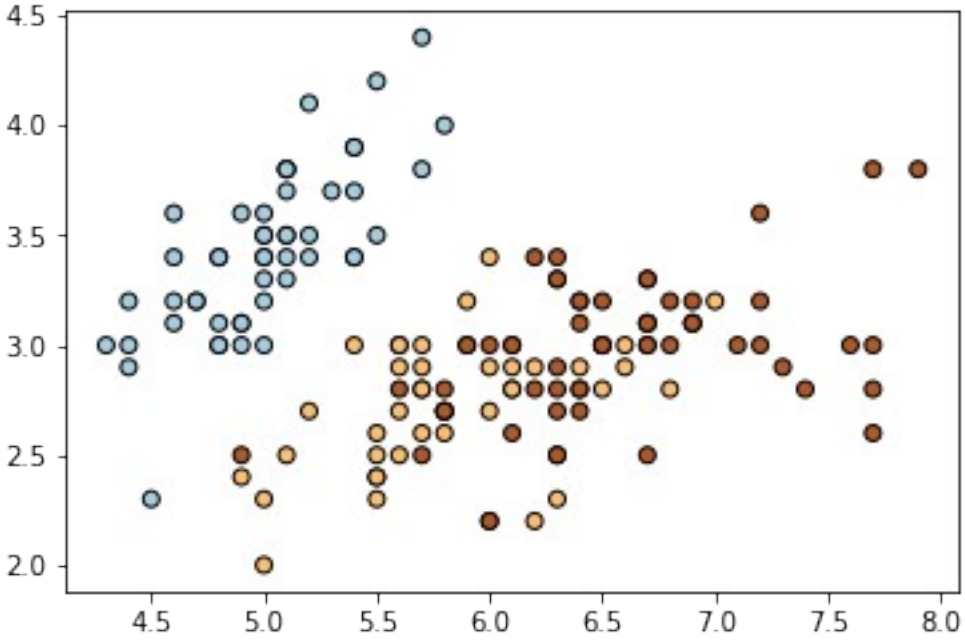
```
0.11755502678948726
```

9b) Logistic Regression

```
import math

def sigmoid(x):
    a = []
    for item in x:
        a.append(1/(1+math.exp(-item)))
    return a

x = np.arange(-10., 10., 0.2)
sig = sigmoid(x)

import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features.
Y = iris.target

x_train, x_test, y_train, y_test =
train_test_split(X,Y,test_size=0.8,random_state=1)

model = LogisticRegression()
```

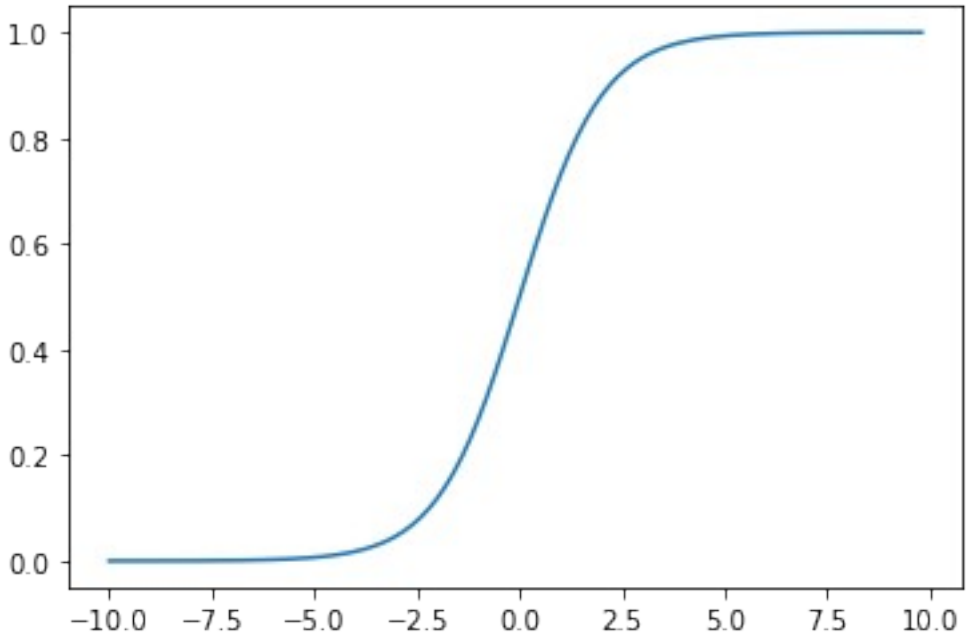
```
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))

plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors="k", cmap=plt.cm.Paired)
#plt.plot(x,sig)
plt.show()
```

```
[[40  0  0]
 [ 0 32  5]
 [ 0 20 23]]
0.7916666666666666
```




```
plt.plot(x,sig)
```

```
[<matplotlib.lines.Line2D at 0x254ec5b9760>]
```

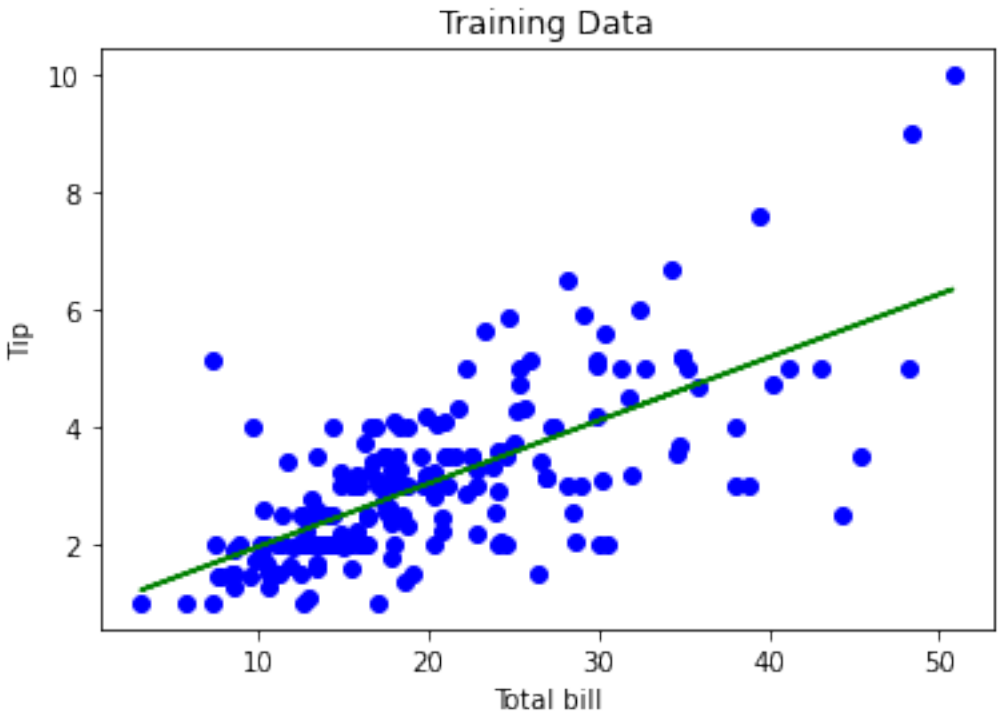


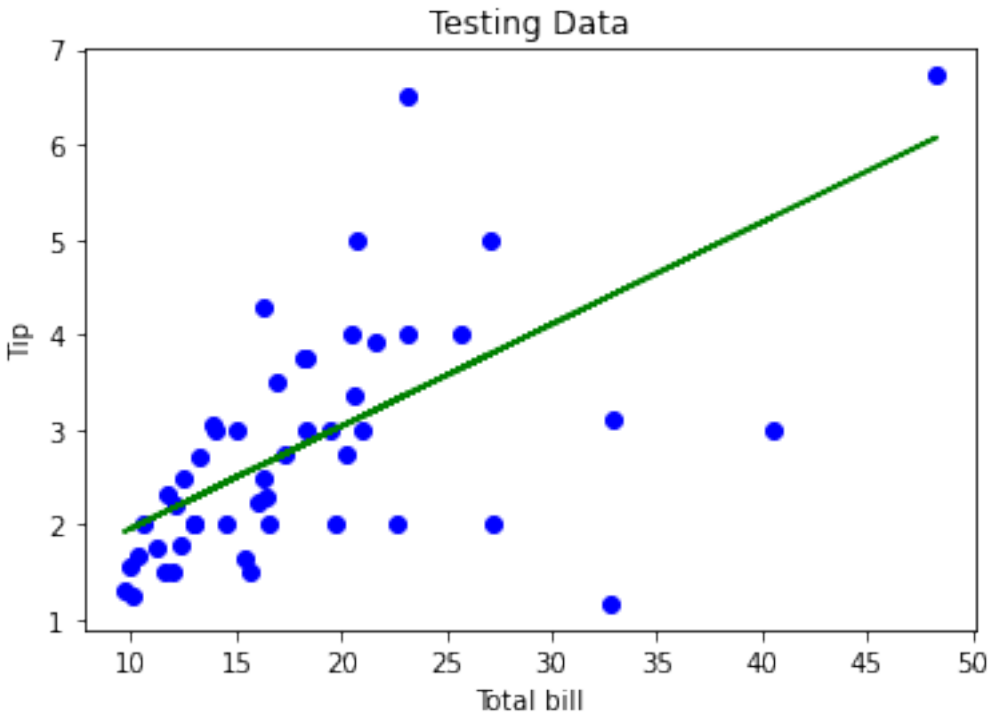
Tej Linear regression

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


df = sns.load_dataset('tips')

X= df[['total_bill']]
y = df[['tip']]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)


# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(X_test)
```

```
# The coefficients
print("Slope: ", regr.coef_)
print("Intercept: ",regr.intercept_)
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))

plt.title("Training Data")
plt.scatter(X_train, y_train, color = "blue")
plt.plot(X_train, regr.predict(X_train), color = "green")
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()

plt.title("Testing Data")
plt.scatter(X_test, y_test, color = "blue")
plt.plot(X_test, regr.predict(X_test), color = "green")
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

```
Slope:  [[0.10751848]]
Intercept:  [0.88394529]
Mean squared error: 1.07
```

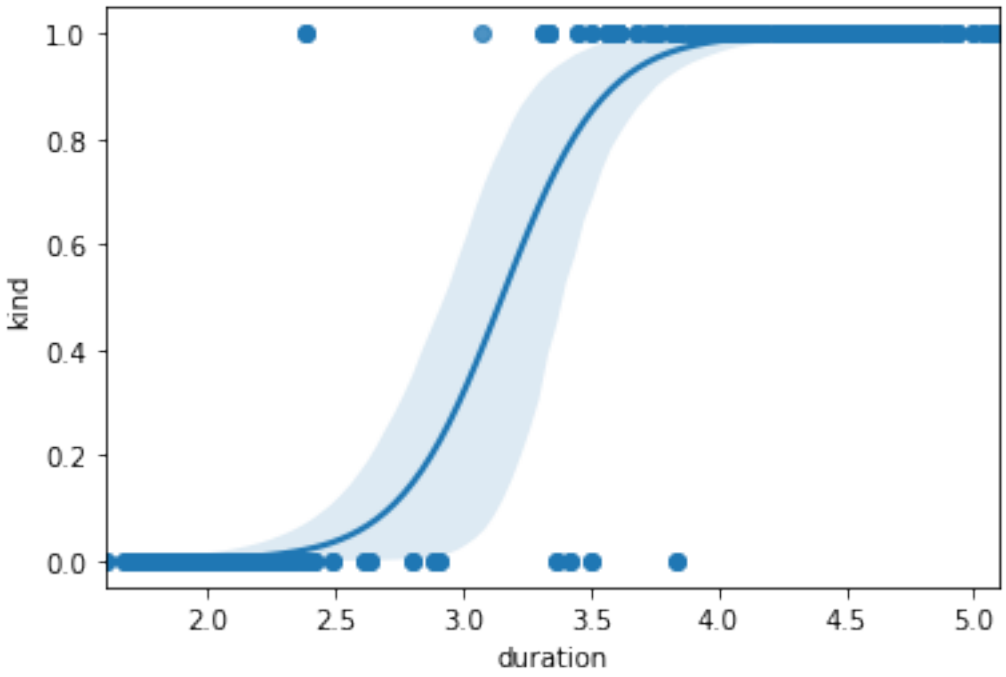




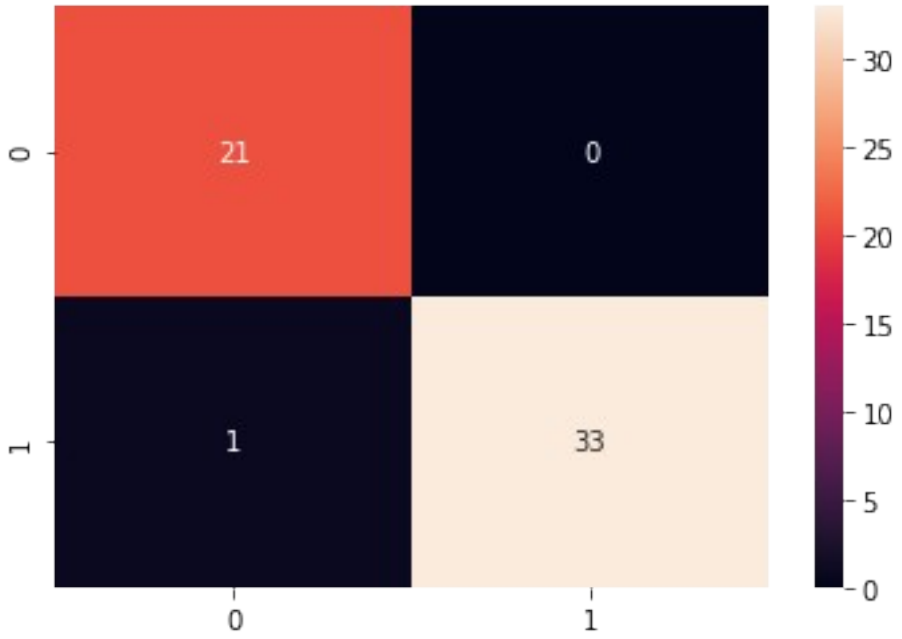
Tej Logistic regression

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")

df = sns.load_dataset('geyser')
df = df.drop(columns='waiting')
df['kind'] = df['kind'].replace(['short'],0)
df['kind'] = df['kind'].replace(['long'],1)

X= np.array(df.duration).reshape(-1,1)
y= np.array(df.kind)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# # Create linear regression object
log = LogisticRegression()

# # Train the model using the training sets
log.fit(X_train, y_train)
```

```
sns.regplot(x='duration', y='kind', data=df, logistic=True)
plt.scatter(X_train,y_train)
plt.show()

sns.heatmap(confusion_matrix(y_test,log.predict(X_test)),annot=True)
print("Accuracy: ")
print(accuracy_score(y_test,log.predict(X_test)))
```




```
Accuracy:
0.9818181818181818
```



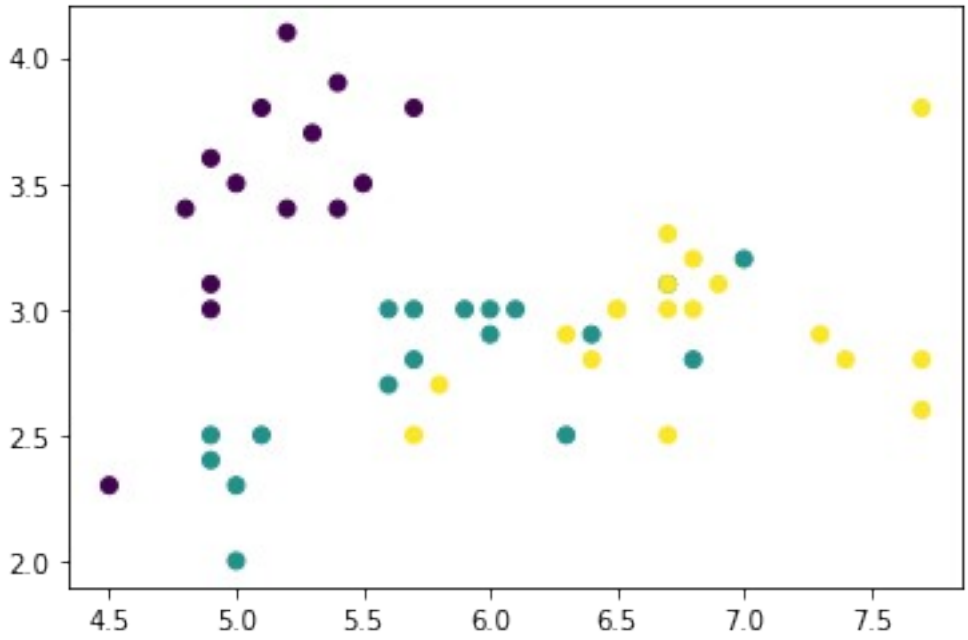
10a) ADAboost Classifier

```
from sklearn import datasets
iris=datasets.load_iris()
x=iris.data
y=iris.target
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
xtr,xt,ytr,yt=train_test_split(x,y,test_size=1/3)
from sklearn.ensemble import AdaBoostClassifier as ab
model=ab()
model.fit(xtr,ytr)
p = model.predict(xt)
print("Classification report:%",classification_report(yt,p))
plt.scatter(xt[:,0],xt[:,1],c=p)
```

```
Classification report:%               precision    recall  f1-score
support

           0       1.00      1.00      1.00        14
           1       0.83      0.94      0.88        16
           2       0.94      0.85      0.89        20

    accuracy                           0.92        50
   macro avg       0.93      0.93      0.93        50
```

```
weighted avg       0.92      0.92      0.92        50
```

```
<matplotlib.collections.PathCollection at 0x254ef3a5df0>
```



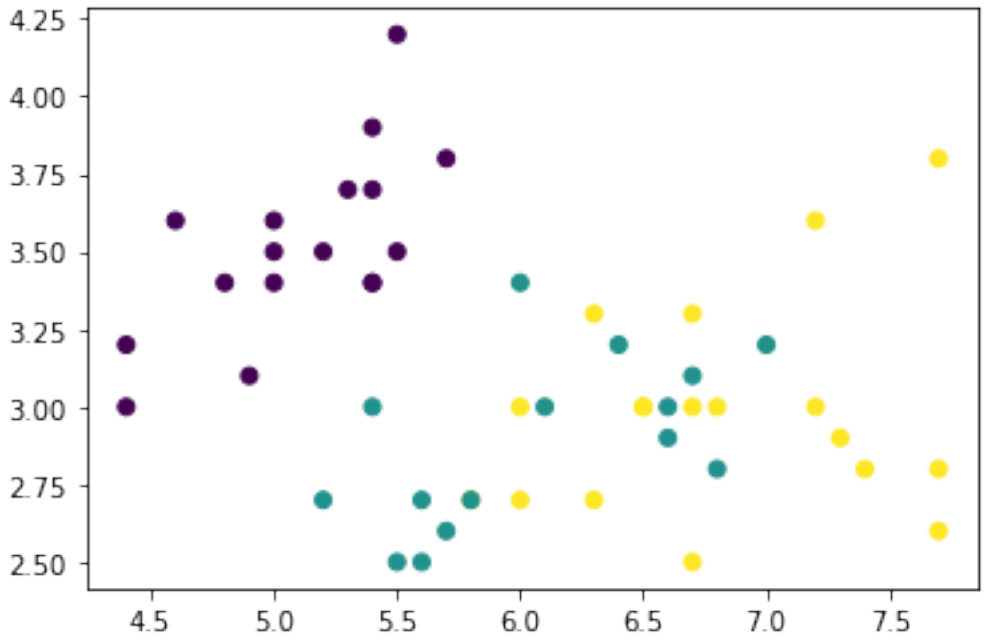
10b) Bayesian Classifier

```
from sklearn import datasets
iris=datasets.load_iris()
x=iris.data
y=iris.target
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
xtr,xt,ytr,yt=train_test_split(x,y,test_size=1/3)
from sklearn.naive_bayes import GaussianNB as nb
model=nb()
model.fit(xtr,ytr)
p = model.predict(xt)
print("Classification report:%",classification_report(yt,p))
plt.scatter(xt[:,0],xt[:,1],c=p)
```

```
Classification report:%               precision    recall  f1-score
support

           0       1.00      1.00      1.00        17
           1       1.00      0.88      0.94        17
           2       0.89      1.00      0.94        16

    accuracy                           0.96        50
```

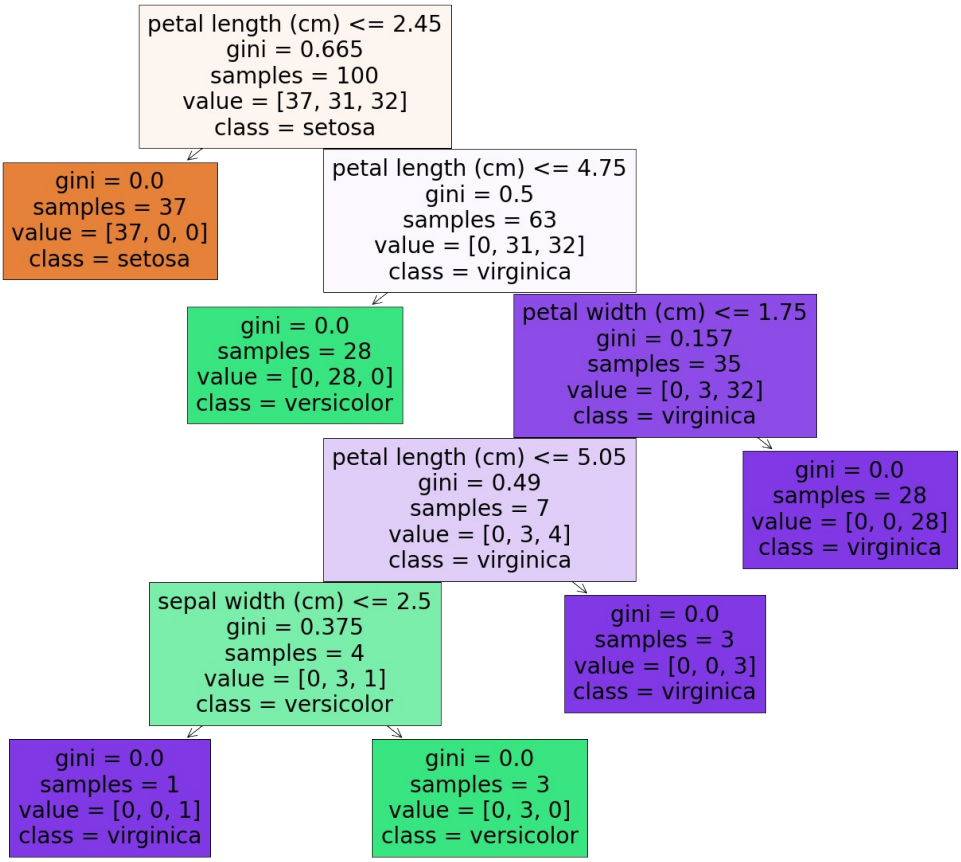
```
   macro avg       0.96      0.96      0.96        50
weighted avg       0.96      0.96      0.96        50
```

```
<matplotlib.collections.PathCollection at 0x254ef55b2b0>
```

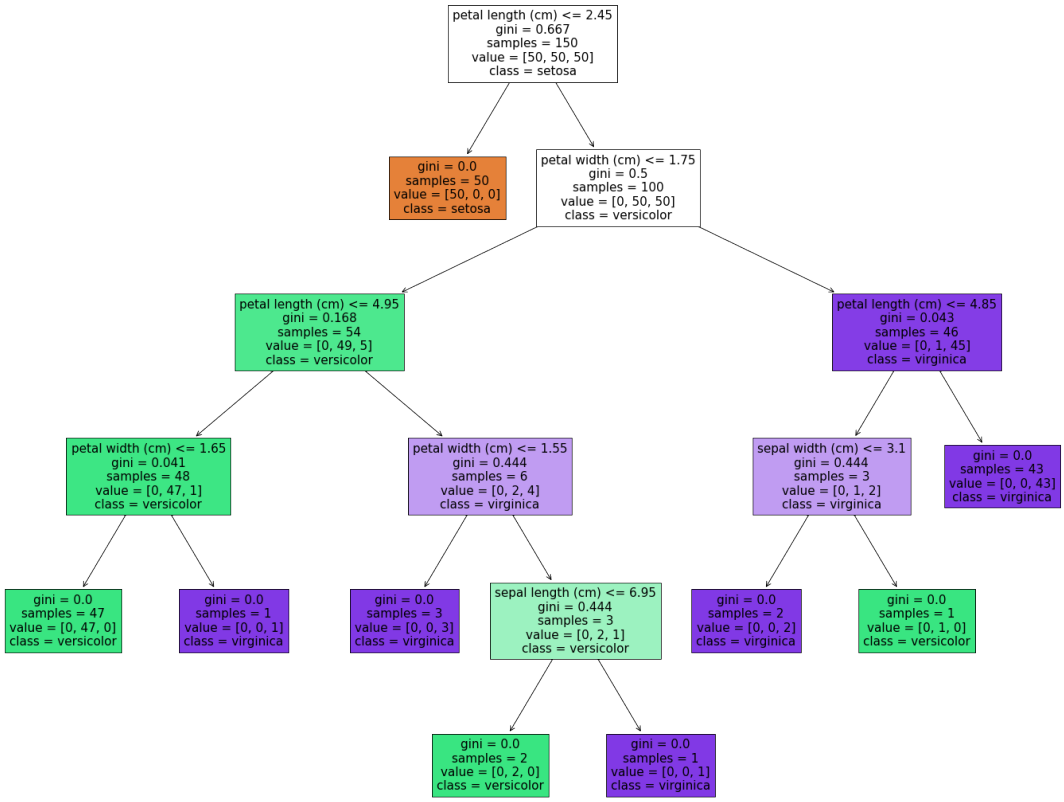


10c) Decision Tree Classifier

```
from sklearn import datasets
iris=datasets.load_iris()
x=iris.data
y=iris.target
xtr,xt,ytr,yt=train_test_split(x,y,test_size=1/3)
from sklearn.tree import DecisionTreeClassifier as f
from sklearn import tree
model=f()
model.fit(xtr,ytr)
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(model,feature_names=iris.feature_names,
class_names=iris.target_names,filled=True)
```




```
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
# Prepare the data data
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Fit the classifier with default hyper-parameters
clf = DecisionTreeClassifier(random_state=1234)
model = clf.fit(X, y)

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
```


petal length (cm) <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm) <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm) <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm) <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm) <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginica
gini = 0.0
samples = 47
value = [0, 47, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
sepal length (cm) <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica